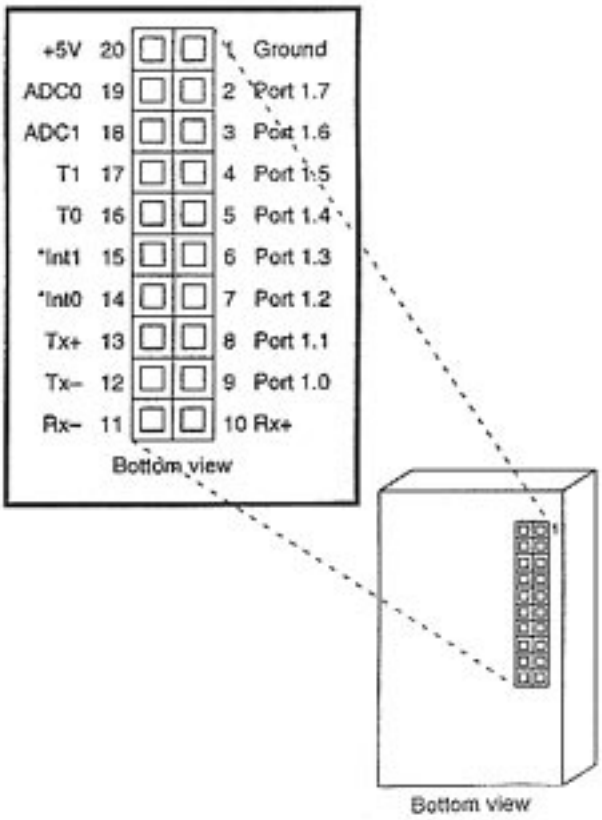
Micromint Chips

DOMINO-1™

Microcomputer/controller with embedded BASIC interpreter

FEATURES

- Small size—complete computer/controller with I/O in less than 1.0 cubic inches (1.1" × 1.79" × 0.5")
- Low power—only 75 mW typical
- Operates on +5 V at 15 mA (typical)
- Communications through RS-232A, RS-422, or RS-485 serial port up to 19.2 kbps; internal on-chip level shifters
- Full floating-point BASIC for easy programming
- On-chip firmware that measures period and frequency and supports two PWM outputs and I²C bus
- 32-KB SRAM for "enter and execute" program testing
- 32-KB EEPROM nonvolatile storage for autostart applications
- 2 × 10 square-pin header carries all signals (mates with ribbon cable or PCB)
- Optional 2-channel, 12-bit ADC, 10k samples/second
- 11.059 MHz system clock
- 2 interrupts and 3 timers
- Parallel I/O—12 bits with 3 shared with ADC and I²C

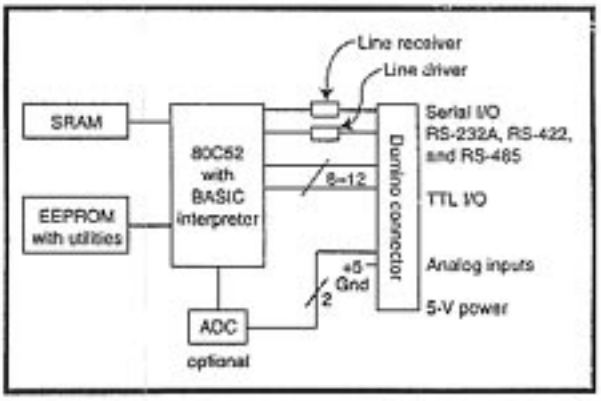


DESCRIPTION

The DOMINO-1 microcontroller is a rugged, miniature controller with a fast, control-oriented, processor masked BASIC interpreter. DOMINO-1 programs can be entirely BASIC or a mixture of BASIC and assembly language routines with a BASIC CALL instruction.

DOMINO-1 is designed to be a 100% stand-alone, low-power, embedded controller which only requires a user to apply power to function. DOMINO-1 is both RS-232A and RS-485 compatible without extra components. Based on a CMOS 80C52 processor, DOMINO-1 provides a ROM-resident BASIC interpreter, 32 KB of static RAM, 32 KB of nonvolatile EEPROM, 12 parallel I/O lines, a 2-channel sample-and-hold 12-bit A/D converter, and integral drivers and receivers for RS-422/485 and RS-232A communications.

Additional firmware enables program calls to directly read frequency and period, set PWM pulse width and duty cycle, communicate with I²C bus peripherals, and save programs to EEPROM that can be autostarted.



Note: Domino-1 was formerly called Domino-52

© 1997 Micromint, Inc.

DOMINO-1™

PINOUT

Ground	Single point digital and analog ground
Port 1.7	TTL I/O bit 7 (available through BASIC; optional ADC uses Port 1.7 as CS input and I²C as CLK)
Port 1.6	TTL I/O bit 6 (available through BASIC; both ADC and I²C use Port 1.6 as Data I/O)
Port 1.5	TTL I/O bit 5 (available through BASIC; optional ADC uses P1.5 as CLK)
Port 1.4	TTL I/O bit 4 (available through BASIC)
Port 1.3	TTL I/O bit 3 (available through BASIC)
Port 1.2	TTL I/O bit 2 (available through BASIC)
Port 1.1	TTL I/O bit 1 (available through BASIC)
Port 1.0	TTL I/O bit 0 (available through BASIC)
Rx+	RS-422/485 noninverted serial (receive pair/rec-xmit pair)
Rx–	RS-422/485/232A inverted serial (receive pair/ rec-xmit pair/receive)
Tx–	RS-422/485/232A inverted serial (transmit pair/rec-xmit pair/transmit)
Tx+	RS-422/485 noninverted serial (transmit pair/rec-xmit pair)
*Int0	TTL interrupt input and general-purpose I/O bit (available through assembly language)**
*Int1	TTL interrupt input and general-purpose I/O bit (available as interrupt through BASIC)**
T0	Serial transmitter disable control, TTL timer/ counter input and general-purpose I/O bit (available through assembly language)**
T1	TTL timer/counter input and general-purpose I/O bit (available through assembly language)**
ADC1	Analog input 1 (0–5 V)
ADC0	Analog input 0 (0–5 V)
+5V	Regulated 5-V input for digital and analog circuitry (analog inputs referenced to this input)

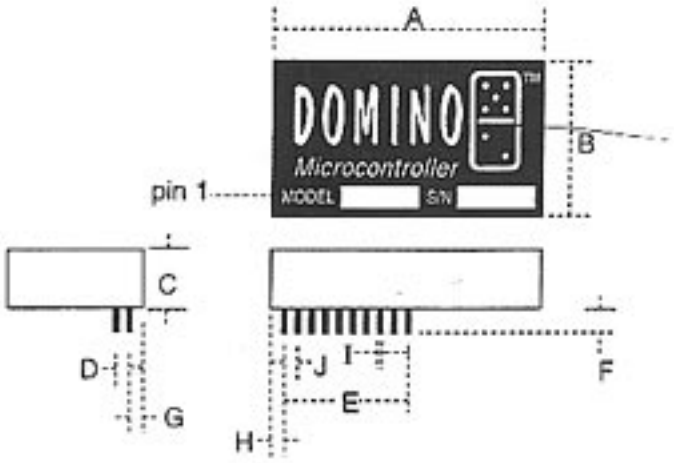
* Triggered on the falling edge
** Refer to Section 5.0 (Controlling I/O Bits Directly)

ABSOLUTE MAXIMUM RATINGS

Operating temperature	Commercial 0°C to +70°C
Industrial	–40°C to +85°C
Storage temperature	–50°C to +125°C
Voltage on Vcc to Vss	0 to +7 V

MECHANICAL AND ENVIRONMENTAL CHARACTERISTICS

Length	1.79 inches
Width	1.1 inches
Height	0.5 inches
Weight	18.5 grams
Operating temperature	0 to +70°C (optional –40 to +85 °C)
Humidity	0 to 100% (noncondensing)



	Inches		Millimeters	
DIM	min	max	min	max
A	1.790	1.800	45.46	45.72
B	1.100	1.110	27.94	28.19
C	0.500	0.510	12.70	12.95
D	0.090	0.110	02.29	02.79
E	0.890	0.910	22.60	23.11
F	0.165	0.165	04.19	04.69
G	0.080	0.100	02.03	02.54
H	0.080	0.100	02.03	02.54
I	0.023	0.027	00.58	00.68
J	0.090	0.110	02.29	02.79

DC ELECTRICAL CHARACTERISTICS

Operating temperature: Ta = 0°C to +70°C
Operating voltage: Vcc = 4.75 V to 5.25 V, Vss = 0.0 V

Characteristic	Minimum	Typical	Maximum	Units	Condition
Supply Voltage (Vcc)	4.75	5.00	5.25	V	
Supply Current (Icc) (RS-422/485 50-Ω termination disabled)		15		mA	
Input Low Voltage (Vil)	–0.5		0.9	V	
Input High Voltage (Vih)	1.9		5.5	V	
Output Low Voltage (Vol)		0.45		V	Iol=1.6 mA
Output High Voltage (Voh)	4.5			V	Ioh=–10 µA
	2.4			V	Ioh=–400 µA

COMMUNICATION LINE DC ELECTRICAL CHARACTERISTICS

Characteristic	Minimum	Typical	Maximum	Units	Condition
Differential Driver Output Voltage			5.0	V	Unloaded See Note 1
RS-422	2.0		5.0	V	R=50 Ω
RS-485	1.5		5.0	V	R=27 Ω
Maximum Receiver Input voltage			±14	V	
ESD Protection		2000		V	

A/D CONVERTER CHARACTERISTICS

Characteristic	Minimum	Typical	Maximum	Units	Condition
Resolution	12			bits	
Linearity Error		±¾		bits	
Offset and Gain Error		±2		bits	
Voltage Reference	5.5	5.0	4.5	V	V_{REF} is Vcc
Analog Input Range		–0.5 to Vcc+0.05		V	See Note 2
Analog Input Impedance		250k		Ω	See Note 3

Note 1: RS-232A is characterized as a ±5-V bipolar signal (as opposed to RS-232C at ±12 V). Drivers and receivers are actually RS-422 and the interface is an RS-423 connection (single ended to differential).

Note 2: Two diodes are tied to each analog input which will conduct when the input voltage is one diode drop below ground or one diode drop above Vcc. To achieve absolute 0–5-V input range requires Vcc to be greater than 4.950 V.

Note 3: The ADC input impedance is a function of clock frequency. The sampling frequency of the DOMINO ADC built-in utility results in a typical impedance of 250 kΩ.

DOMINO-1™

1.0 PROGRAMMING CHARACTERISTICS

DOMINO-1 is a complete computer/controller in one tiny package. The embedded BASIC interpreter and firmware provide the user with a direct means to enter and save an autostarting control program without expensive development tools. Such powerful advantages facilitate completing a programming task in record time. You can write, test, and save code in nonvolatile storage directly on DOMINO-1.

The friendly, control-oriented BASIC command set allows easy access to the integrated digital and analog I/O functions. Conversion calculations are a breeze thanks to BASIC's floating-point number crunching. Because of the power of a high-level language such as BASIC, useful programs often take less than a dozen programming statements. Nonetheless, DOMINO-1 has over 30 KB of space reserved for your application code and the utilities.

Even though DOMINO-1 is optimized for BASIC programs, assembly language programs are easily accommodated as callable routines. A DOMINO-1 application program can be all BASIC, BASIC with callable assembly language routines, or virtually all assembly language with the only BASIC command being an introductory CALL.

DOMINO-1 contains all the communication interface hardware. It can be used standalone to monitor analog and digital inputs and to provide control outputs directly to machine or network interfaces. When connected serially, DOMINO-1 can serve as a remote device, reporting monitored conditions to your PC or receiving commands to control external components. If multiple DOMINO-1s are networked with a master PC or another DOMINO-1, multidrop units can share information collected throughout the network.

2.0 MEMORY MAP

The 64-KB memory is based on an 8051 microcontroller's memory structure. The upper 32 KB is devoted to ROM and the lower 32 KB to RAM.

2.1 DEVELOPMENT MODE

When you're in the development mode, the lower 32 KB of memory is used as temporary storage for BASIC's programs, variables, and jump vectors.

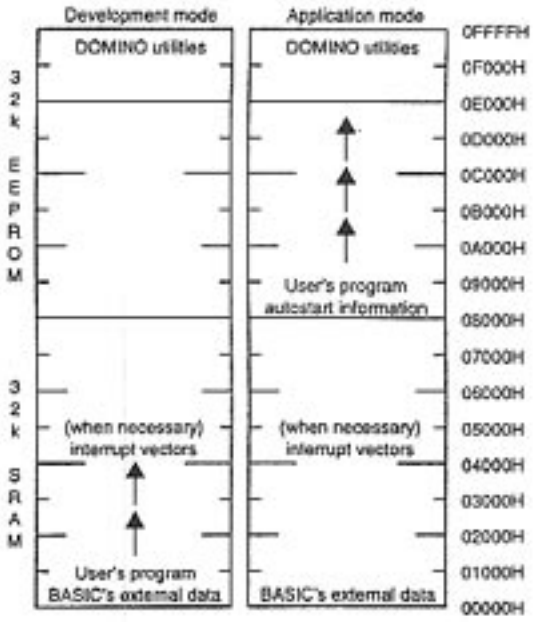
The development mode is used to test and debug BASIC programs. The top of the upper ROM space holds the utilities which are callable functions complementing BASIC's floating-point commands.

2.2 APPLICATIONS MODE

Finished BASIC programs are saved in the upper 32 KB of nonvolatile ROM space along with the utilities. BASIC programs can be autoexecuted on powerup or reset.

The lower 32 KB of RAM space is used for storage of temporary variables and jump vectors.

Memory management



3.0 BASIC INSTRUCTION SET

Command	Function
RUN	Execute a program
CONT	Continue after a stop or Control-C
LIST	List program to the console device
LIST#	List program to serial printer port (P1.7)
NEW	Erase the program stored in RAM
NULL	Set null count after carriage return/line feed
RAM	Evoke RAM mode, current program in read/write memory
ROM	Evoke ROM mode, current program in ROM/EPROM
XFER	Transfer a program from ROM/EPROM to RAM

Statement	Function
BAUD	Set data-transmission rate for line-printer port
CALL	Call assembly-language program
CLEAR	Clear variables, interrupts, and strings
CLEARS	Clear stacks
CLEARI	Clear interrupts
CLOCK1	Enable real-time clock
CLOCK0	Disable real-time clock
DATA	Data to be read by READ statement
READ	Read data in DATA statement
RESTORE	Restore READ pointer
DIM	Allocate memory for arrayed variables
DO	Set up loop for WHILE or UNTIL
UNTIL	Test DO loop condition (loop if false)
WHILE	Test DO loop condition (loop if true)
END	Terminate program execution
FOR-TO-[STEP]	Set up FOR...NEXT loop
NEXT	Test FOR...NEXT loop condition
GOSUB	Execute subroutine
RETURN	Return from subroutine
GOTO	GOTO program line number
ON GOTO	Conditional GOTO
ON GOSUB	Conditional GOSUB
IF-THEN-[ELSE]	Conditional test
INPUT	Input a string or variable
LET	Assign a variable or string a value (LET is optional)
ONERR	ONERR or GOTO line number
ONTIME	Generate an interrupt when time is equal to or greater than ONTIME argument; line number is after comma
ONEX1	GOSUB to line number following ONEX1/ when INT1 pin is pulled low
PRINT	Print variables, strings, or literals. P. is shorthand for print
PRINT#	Print to serial printer port (P1.7)
PH0.	Print hexadecimal mode with zero suppression
PH1.	Print hexadecimal mode with no zero suppression
PH0.#	PH0.# to serial printer port (P1.7)
PH1.#	PH1.# to serial printer port (P1.7)
PUSH	Push expressions on argument stack
POP	Pop argument stack to variables
PWM	Pulse-width modulation
REM	Remark
RETI	Return from interrupt
STOP	Break program execution
STRING	Allocate memory for strings
UI1	Evoke user console input routine
UI0	Evoke BASIC console input routine
UO1	Evoke user console output routine
UO0	Evoke BASIC console output routine

Operator	Function
CBY( )	Read program memory
DBY( )	Read/assign internal data memory
XBY( )	Read/assign external data memory
GET	Read console
IE	Read/assign IE register
IP	Read/assign IP register
PORT1	Read/assign I/O port 1 (P1)
PCON	Read/assign PCON register
RCAP2	Read/assign RCAP2 (RCAP2H:RCAP2L)
T2CON	Read/assign T2CON register
TCON	Read/assign TCON register
TMOD	Read/assign TMOD register
TIME	Read/assign real-time clock
TIMER0	Read/assign TIMER0 (TH0:TL0)
TIMER1	Read/assign TIMER1 (TH1:TL1)
TIMER2	Read/assign TIMER2 (TH2:TL2)
+	Addition
/	Division
**	Exponentiation
*	Multiplication
-	Subtraction
.AND.	Logical AND
.OR.	Logical OR
.XOR.	Logical exclusive OR

Stored Constant

PI	PI - 3.1415926

Operators-Single Operand

ABS( )	Absolute value
NOT( )	One's complement
INT( )	Integer
SGN( )	Sign
SQR( )	Square root
RND	Random number
LOG( )	Natural log
EXP( )	'e' (2.7182818) to the X
SIN( )	Returns the sine of argument
COS( )	Returns the cosine of argument
TAN( )	Returns the tangent of argument
ATN( )	Returns the arctangent of argument

Utility Calls (executed as CALL (address))

PROG	Save the current program in EEPROM
PROG1	Save data-transmission-rate information in EEPROM
PROG2	Save data-transmission-rate information in EEPROM and execute program after reset
PROG3	Save data-transmission-rate information in EEPROM and saves MTOP
PROG4	Save data-transmission-rate information in EEPROM and execute program after reset
ADC	read 0–5-V input on AD0 or AD1, measurement returned as floating-point value
PWM	continuous background PWM tasking
FREQ	measurement of TTL input frequency
PERIOD	measurement of TTL input period
I²C	communication with external I²C peripherals

4.0 DOMINO FUNCTION CALL PROCEDURES (REV 1.00)

Micromint has included additional utilities with its built-in BASIC interpreter. Not only do you have the power of a full floating-point BASIC, but you also have extra functions to help make your application extremely easy to produce. The added functions include analog measurement, dual PWM outputs, dual period/frequency input measurements, I²C bus compatibility (e.g., LCD output and keypad input), and program storage in EEPROM for autostarting your application on power-up. These functions are written in assembler to be extremely fast. They are simple to use straight from BASIC.

ADC	read 0–5-V input on AD0 or AD1, measurement returned as floating-point value
PWM	continuous background PWM tasking
FREQ	measurement of TTL input frequency
PERIOD	measurement of TTL input period
PROG1–4	autoexecutable program storage into nonvolatile EEPROM
I²C	communication with external I²C peripherals

These function calls are loaded into the EEPROM above BASIC program storage by a utility loader. The DOMINO-1 firmware is preloaded at the factory prior to shipment. Should it be accidently erased or need to be revised, it can be reprogrammed using the bootstrap loader diskette included in the DOMINO-1 development software package.

DOMINO-1 is potentially reprogrammable even while soldered in an end-use application. This reprogramming reduces obsolescence, making it possible for a user to upgrade current DOMINO-1 stock with the latest enhancements.

4.1 12-BIT ANALOG-DIGITAL CONVERSION

Syntax: CALL (address)
POP (variable)

Function: The CALL initiates an analog-to-digital conversion. The result is presented on the stack to be POPed by the user.

Mode: Command, Run

Use: [single-ended channel 0]
CALL 0F000H
POP (variable)

[single-ended channel 1]
CALL 0F008H
POP (variable)

[differential +/-]
CALL 0F010H
POP (variable)

[differential -/+]
CALL 0F018H
POP (variable)

[single-ended channel 1 & 0]
CALL 0F020H
POP (variable),(variable)

Description: The processor's Port1 pins (P1.7 *CS, P1.6 Data, and P1.5 CLK) are used to access either the internal LTC1298 (DOMINO-1A) or an externally connected LTC1298 (DOMINO-1). The LTC1298 offers a number of different connection configurations. Two ADC input channels are available when each is a single-ended measurement (referenced to ground). Alternatively, these channels can be used as a single differential input (neither is ground referenced but there is no greater than 5 V between them). Channel 0 is +input, channel 1 is –input.

Related Topics: 8-bit A/D Conversion

Error Presentation: No errors presented. A CALL made to a nonexistent ADC still returns a value on the stack, albeit one of no meaning.

Example:

```
10   PRINT "This program prints an A/D conversion"
20   PRINT " from two single-ended inputs: Channel
      1 & 0"
30   INPUT"Measure and enter your VCC voltage
      (e.g., 5.12)"P
40   CALL 0F020H: REM THE FUNCTION CALL
50   POP V1,V0: REM GETTING THE RESULTS
60   PRINT USING(0),"Channel 1's conversion count
      is",V1
70  PRINT " and the calculated voltage is",
80  PRINT USING(0.000), V1 * (P/4096)," volts"
90   PRINT USING(0),"Channel 0's conversion count
      is",V0
100   PRINT " and the calculated voltage is",
110   PRINT USING(0.000), V0 * (P/4096)," volts"
120  PRINT "Hit a <cr> to make another conversion"
     : PRINT
130  IF (GET=0) THEN GOTO 130 ELSE GOTO 40

READY
>RUN
```

Program Output:

```
This program prints an A/D conversion
 from two single-ended inputs: Channel 1 & 0
Measure and enter your VCC voltage (e.g., 5.12) ?
4.95
Channel 1's conversion count is 254
 and the calculated voltage is 0.310 volts
Channel 0's conversion count is 1259
 and the calculated voltage is 1.521 volts
Hit a <cr> to make another conversion
```

4.2 8-BIT ANALOG-DIGITAL CONVERSION

Syntax: CALL (address)
POP (variable)

Function: The CALL initiates an analog-to-digital conversion. The result is presented on the stack to be POPed by the user.

Mode: Command, Run

Use: [single-ended channel 0]
CALL 0F080H
POP (variable)

Description: The processor's Port1 pins (P1.7 *CS, P1.6 Data, and P1.5 CLK) are used to access an externally connected ADC0831 (DOMINO-1). The ADC0831 offers a single-ended measurement (referenced to ground).

Related Topics: 12-bit A/D Conversion

Error Presentation: No errors presented. A CALL made to a nonexistent ADC still returns a value on the stack, albeit one of no meaning.

Example:

```
10   PRINT "This program prints an A/D conversion"
20   PRINT " from a single-ended input on Channel
      0"
30   INPUT"Measure and enter your VCC voltage
      (e.g., 5.12)"P
40   CALL 0F080H: REM THE FUNCTION CALL
50   POP V0:  REM GETTING THE RESULTS
60   PRINT USING(0),"Channel 0's conversion count
      is",V0
70   PRINT " and the calculated voltage is",
80   PRINT USING(#.###), V0 * (P/256)," volts"
90   PRINT "Hit a <cr> to make another conversion"
      : PRINT
100  IF (GET=0) THEN GOTO 100 ELSE GOTO 40

READY
>RUN
```

Program Output:

```
This program prints an A/D conversion
 from a single-ended input on Channel 0.
Measure and enter your VCC voltage (e.g., 5.12) ?
4.95.
Channel 0's conversion count is 54
 and the calculated voltage is 1.044 volts.
Hit a <cr> to make another conversion
```

4.3 PULSE-WIDTH MODULATION

Syntax: PUSH (On time),(Off time),(Duration)
CALL (address)

where variable (On time) = integer 150–65535 counts
(Off time) = integer 150–65535 counts
(1 count = 1.085 µs)
(Duration) = integer 0–65535 cycles
(0 = continuous)
and given that 1%–99% duty cycle pulses up to 60 Hz
50% duty cycle pulses up to 3 kHz

Function: Defines the on time (high), off time (low), and duration (# of complete cycles) for a PWM output signal. It also starts the PWM output. A duration of zero means continuous output. Two separate PWM outputs are available *INT0 uses TIMER0 and T1 uses TIMER1.
WARNING: Using either of these functions disables any other function using TIMER0, TIMER1, or INTERRUPT0.

BASIC commands using TIMER0: CLOCK1
TIMER1: PWM, LIST#, PRINT#
INTERRUPT 0: none

Mode: Command, Run

Use: [start PWM0 using TIMER0 and *INT0 as output]
PUSH 500,1500,0
CALL 0F060H

```
[stop PWM0 immediately]
        CALL 0F068H

    [start PWM1 using TIMER1 and T1 as output]
        PUSH 500,1500,0
        CALL 0F064H

    [stop PWM1 immediately]
        CALL 0F06CH
```

Description: Using the PWM function requires MTOP to be set to 3FFFH (although the function call sets this, the user should be aware that any variable used prior to this call is destroyed unless MTOP is preset to 3FFFH at the beginning of a program).

TIMER interrupt vector locations (400BH–400DH and 401BH–401DH) and on time, off time, and duration values storage locations (4200H–420BH) are set up in RAM. The PWM function call sets up the TIMER counts alternating between the on-time value and the off-time value on each TIMER overflow until the duration value has been decreased to zero.

A separate function call can be made at any time to immediately shut down the PWM. Each on- and off-time count defined is the number of 1.085-µs tics the routine delays before changing state. The minimum count is 150 (150 × 1.085 µs) or 163 µs. The max count is 65535 or 71 ms.

Related Topics: PWM (BASIC command) The BASIC-1 Interpreter's PWM command halts execution of the BASIC program while it is being executed. PWM0 and PWM1 function calls do NOT halt the execution of the BASIC program, but it becomes a background task.

Error Presentation: No error are reported although any BASIC command which uses the timers is disabled (see Function description above).

Example: This example sets up both PWM outputs with continously varying 1–99% duty cycles.

```
10   FOR Y=150 TO 14700 STEP 300
20   PUSH Y
30   PUSH 15000-Y
40   PUSH 0
50   CALL 0F060H
60   PUSH 15000-Y
70   PUSH Y
80   PUSH 0
90   CALL 0F064H
91   FOR Z=1 TO 50: NEXT Z
100  NEXT Y
110  FOR Y=14700 TO 150 STEP -300
120  PUSH Y
130  PUSH 15000-Y
140  PUSH 0
150  CALL 0F060H
160  PUSH 15000-Y
170  PUSH Y
180  PUSH 0
190  CALL 0F064H
191  FOR Z=1 TO 50: NEXT Z
200  NEXT Y
210  GOTO 10

READY
>RUN
```

4.4 PERIOD AND FREQUENCY

Syntax: CALL (address) [Start measurement]

```
    CALL (address) [Retrieve result]
    POP (variable)
```

where variable (period count) = integer 0–65535
(0 = measurement started)
(1 = measurement in process)
(2 = overflow occured—signal too slow)
(60–65535 = counts between negative edges)
(1 count = 1.085 µs)
(period = 65 µs – 71 ms)
(frequency = 15 kHz – 15 Hz)

Function: The start measurement function call sets up the edge-triggered input interrupts and timers used to measure the period between two sucessive input edges. Two separate input signals can be measured. Input *INT0 uses interrupt 0 and timer0 and input *INT1 uses interrupt 1 and timer1.
WARNING: Using either of these inputs disables any other function using the interrupts or timers. The timers and interrupts may again be used after the function calls are complete.

BASIC commands using:
TIMER0: CLOCK1
INTERRUPT0: none
TIMER1: PWM
LIST#, PRINT# INTERRUPT1: ONEX1

Mode: Command, Run

Use:
```
[Start a measurement on input *INT0]
    CALL 0F070H

[Start a measurement on input *INT1]
    CALL 0F074H

[Retrieve a measurement on input *INT0]
    CALL 0F078H
    POP P

[Retrieve a measurement on input *INT1]
    CALL 0F07CH
    POP P
```

Descripton: Using the PERIOD/FREQUENCY function requires MTOP to be set to 3FFFH (although the function call sets this, the user should be aware that any variable used prior to this call is destroyed unless MTOP is preset to 3FFFH at the beginning of a program).

External interrupt vector locations (4003H–4005H and 4013H–4015H), TIMER interrupt vector locations (400BH–400DH and 401BH–401DH) and period count storage locations (420CH–420FH) are set up in RAM. The start period measurement function call initializes the INTERRUPT and TIMER. The retrieve measurement function call passes the measurement status back to the user via the stack. The status is indicated as follows:

POPed Value	Meaning
0	waiting for input
1	measurement in process
2	overflow (input too slow or nonexistent)
other	counts in 1.065 µs intervals.

Related Topics: none

Error Presentation: No errors reported (see Function description above).

Example:

```
10   PRINT "This program lets you use the
      frequency function"
20   PRINT "Apply the TTL frequency to pin *INT0
      and/or *INT1"
30   CALL 0F070H: REM PERIOD COUNT ON *INT0
      FUNCTION CALL
40   CALL 0F074H: REM PERIOD COUNT ON *INT1
      FUNCTION CALL
50   CALL 0F078H: REM RETRIEVE COUNT ON *INT0
      FUNCTION CALL
60   POP PC0: REM GET THE PERIOD COUNT
70   IF (PC0=2) THEN PRINT "The frequency is too
      low on *INT0": GOTO 130
80   IF (PC0=0.OR.PC0=1) THEN GOTO 50
90   P=PC0*1.085: REM PERIOD TIME CALCULATION
      FROM COUNT VALUE
100  PRINT "The period on *INT0 is",P," µs. The
      frequency is",
110  F=1/P*1000000: REM FREQUENCY CALCULATION
      FROM PC VALUE
120  PRINT F," Hz"
130  CALL 0F07CH: REM RETRIEVE COUNT ON *INT1
      FUNCTION CALL
140  POP PC1: REM GET THE PERIOD COUNT
150  IF (PC1=2) THEN PRINT "The frequency is too
      low on *INT1": GOTO 210
160  IF (PC1=0.OR.PC1=1) THEN GOTO 130
170  P=PC1*1.085: REM PERIOD TIME CALCULATION
      FROM COUNT VALUE
180  PRINT "The period on *INT1 is",P," µs. The
      frequency is",
190  F=1/P*1000000: REM FREQUENCY CALCULATION
      FROM PC VALUE
200  PRINT F," Hz"
210 PRINT "Hit a <cr> to take another sample":
     PRINT
220  IF (GET=0) THEN 220 ELSE GOTO 30

READY
>RUN
```

Program Output:

```
This program lets you use the frequency
function
Apply the TTL frequency to pin *INT0 and/or *INT1
The frequency is too low on *INT0
The period on *INT1 is 2164.575 µs. The frequency is
461.9 Hz
Hit a <cr> to take another sample
```

4.5 PROGRAM EEPROM (PROGx)

Syntax: CALL (address)

Function: The BASIC program residing in RAM and the appropriate header information (autostarting, baud rate, and MTOP) is stored in EEPROM.

Mode: Command

Use: [PROG]
CALL 0FF00H

[PROG1]
CALL 0FF08H

[PROG2]
CALL 0FF10H

[PROG3]
CALL 0FF18H

[PROG4]
CALL 0FF20H

Description: These function calls act just like the BASIC-1 PROG commands. The BASIC commands are written for EPROM. The EEPROM used here requires a different programming algorithm. The PROG function call replaces the BASIC PROG command and saves only the program. The remaining PROGx function calls are similar in function to the BASIC commands, but the function calls all save the program and the startup characteristics in a single call.

Related Topics: PROG, PROG1, PROG2, PROG3, and PROG4 (all BASIC commands)

Error Presenation:

ABORT, PROGRAMMING ERROR!
[EEPROM life exceeded]
ABORTED, ILLEGAL ACCESS ATTEMPT!
[storage space exceeded]
ABORTED, UNKNOWN RESULT CODE!
[unknown error]
ABORTED, NOTHING TO PROGRAM!
[no program in RAM]

Example:

[Type in your program:]

```
10   PRINT"Hello World!"
```

[Type RUN to verify it executes properly:]

```
RUN
Hello World!

>READY
```

[Now type the function call for PROG2:]

```
CALL 0FF10H
```

[you should see:]

```
STORING PROGRAM...
PROGRAM STORED!
```

[Whenever the power is disconnected and reconnected you should see:]

```
Hello World!
```

4.6 I²C BYTE TRANSFERS

Syntax: [send registered BYTE]
PUSH (slave address * 100H + slave register)
PUSH (8-bit value)
CALL (address)
POP (16-bit value)

[retrieve registered BYTE]
PUSH (slave address * 100H + slave register)
CALL (address)
POP (16-bit value)

[send BYTE]
PUSH (slave address * 100H + 8-bit value)
CALL (address)
POP (16-bit value)

[retrieve BYTE]
PUSH (slave address * 100H)
CALL (address)
POP (16-bit value)

Function: Communication is attempted with an I²C device. An 8-bit value is passed to and from the device.

Mode: Command, Run

Use: where A=slave address
R=slave register
V=value to send
C=value retrieved

[send registered BYTE]
PUSH A*100H+R,V
CALL 0F128H
POP C

[retrieve registered BYTE]
PUSH A*100H+R
CALL 0F12CH
POP C

[send BYTE]
PUSH A*100H+V
CALL 0F120H
POP C

[retrieve BYTE]
PUSH A*100H
CALL 0F124H
POP C

Description: The address and register of the I²C slave device is passed on the stack. If an 8-bit value is to be sent, it too must be pushed onto the stack. A call is then made to send a message using the I²C bus (P1.7 CLK and P1.6 DATA). The routine returns a 16-bit value to the user on the stack. The upper byte of the returned value is zero (000xxH) if the transfer was sucessful. Otherwise, it is set to all 1s (0FFxxH). If the function was to retrieve a byte, it is in the lower 8 bits of the 16-bit return.

Related Topics: I²C Beeper, I²C Keypad, and I²C LCD

Error Presentation: The upper 8 bits of the received byte are masked to all 1s if the transmission is unsucessful or 0s if all is OK.

Example:

```
10   PRINT"Turn ON the beeper"
20   A=046H
30   V=0DFH
40   PUSH A*100H+V
50   CALL 0F120H
60   POP C
70   IF (C<>0) THEN GOTO 180
80   PRINT"Hit a key to turn it OFF"
90   G=GET
100  IF (G=0) THEN GOTO 90
110  V=0FFH
120  PUSH A*100H+V
130  CALL 0F120H
140  POP C
150  IF (C<>0) THEN GOTO 180
160  PRINT"Now it's OFF"
170  END
180  PRINT"Error in I2C communications"
190  END
```

4.6.1 I²C BEEPER

This function call assumes you're using a Philips/Signetics PCF8574 I²C 8-bit I/O expander with the slave address 01000110 and piezobeeper on bit 5.

Syntax: CALL (address)

Function: Bit 5 of the slave I/O expander is momentarily set low to produce a short burst from an attached piezo-beeper.

Mode: Command, Run

Use: CALL 0F110H

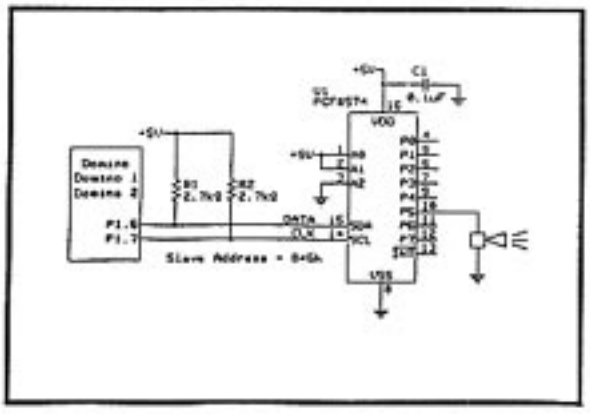
Description: The preassigned I²C slave address 46H is written to with a value of DFH to turn off bit 5. After a short delay, a value of FFH is sent to turn bit 5 back on. The output bit can sink 25 mA of current for, in this case, a piezoelectric beeper.

Related Topics: I²C Keypad, I²C LCD

Error Presentation: none

Example:

```
10   PRINT"Beep the beeper"
20   CALL 0F110H
30   PRINT"That's it!"
40   END
```



4.6.2 I²C KEYPAD

This function call assumes you're using a Philips/Signetics PCF8574 I²C 8-bit I/O expander with the slave address 01000110 and input bits 0–4 from 74C922 data bits, input bit 7 from Q of 74HC74 clocked from grounded D by DAV from 74C922, and output bit 6 to set the 74HC74)

Syntax: [initialize]
CALL (address)

[hook into UI0/1 BASIC command]
CALL (address)

Function: The slave I/O expander is initialized by toggling output bit 6 low and high to set the 74HC74. This clears the DAV latch. The hook routines can be implemented to install the keypad as the alternate console input device.

Mode: Command, Run

Use: [initialize]
CALL 0F100H

[hook into UI0/1 BASIC command]
CALL 0F108H

Description: To initialize the keypad, the preassigned I²C slave address 46H is written to with a value of BFH to turn off bit 6. A value of FFH turns bit 6 back on. The output bit sets a 74HC74 latch clearing any DAV which may have clocked the grounded D input. The DAV signal is generated by a 74HC922 whenever a key is pressed. The keyboard can be read through an I²C read BYTE routine. If bit 7 is low, then the lower 4 bits contain keypad data. The user must reinitialize the keyboard (clear the DAV) after each key read.

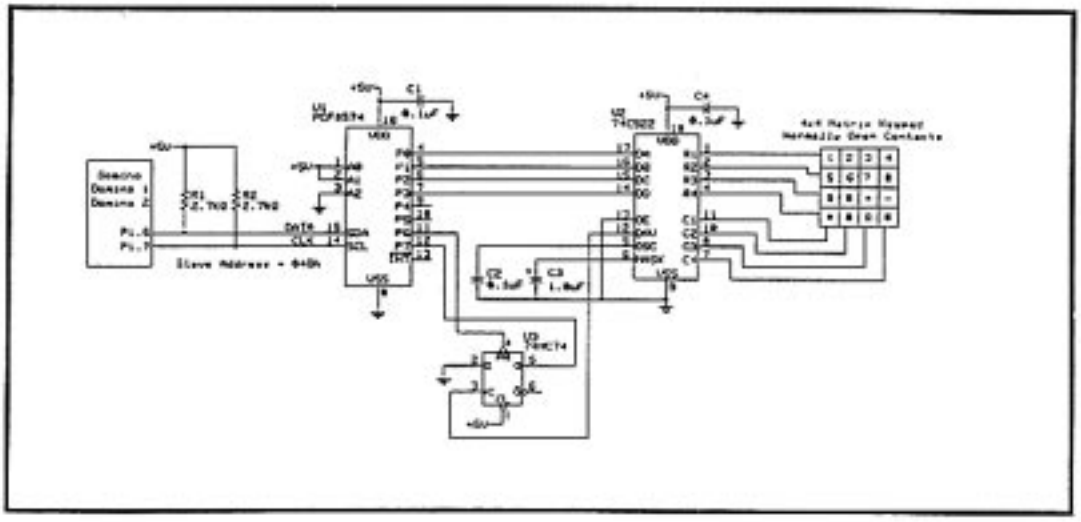
Alternatively, and much easier, the second function hooks the keypad into the alternate console input device. Using the hook function requires MTOP to be set to 3FFFH (although the function call sets this, the user should be aware that any variable used prior to this call is destroyed unless MTOP is preset to 3FFFH at the beginning of a program). Custom console input vector locations (4033H–4035H) and custom console status check vector locations (4036H–4038H) are set up in RAM.

Related Topics: I²C BYTE transfers, I²C LCD, UI0 and UI1 (BASIC commands)

Error Presentation: none

Example:

```
10   MTOP=03FFFH
20   PRINT"Initialize the Keypad"
30   CALL 0F100H
40   PRINT"Hook into the alternate console input"
50   CALL 0F108H
60   PRINT"Hit a key on the keyboard to swap to
      keypad input"
70   G=GET
80   G=GET: IF (G=0) THEN GOTO 80
90   PRINT: PRINT"Now swapping to keypad input"
100  UI1: REM CHANGE TO ALTERNATE CONSOLE INPUT
110  PRINT"Hit a key on the keypad to swap to
      keyboard input"
120  G=GET
130  G=GET: IF (G=0) THEN GOTO 130
140  PRINT: PRINT"Now swapping to keyboard input"
150  UI0: REM CHANGE TO PRIMARY CONSOLE INPUT
160  GOTO 60
```



4.6.3 I²C LCD

This function call assumes you're using a Philips/Signetics PCF8574 I²C 8-bit I/O expander with the slave address 01000010 and output bit 0–3 to the (LM034—4x20) LCD data bits 4–7, output bit 4 to the LCD RS pin, and output bit 5 to the LCD E pin).

Syntax: [initialize]
CALL (address)
[clear display & home cursor]
CALL (address)
[display $(0) string]
CALL (address)
[display a character]
CALL (address)
[hook into UI0/1 BASIC command]
CALL (address)

Function: The LCD must be initialized through the slave I/O expander. This sets up the LCD in nybble mode with a 4 × 7 character matrix and invisible cursor. The LCD is cleared and the cursor set to row1 column 1. Once initialized, the LCD can be cleared and cursor sent home at any time. The first string, $(0), can be directed to the LCD without using console redirection (UO1).

If you choose to use console redirection (UO1), the characters are handled one at a time. Console redirection can be invoked once the hooks are installed for the BASIC UI1 command.

Mode: Command, Run

Use: [initialize]
CALL 0F030H
[clear display & home cursor]
CALL 0F038H
[display $(0) string]
CALL 0F040H
[display a character]
CALL 0F048H
[hook into UI0/1 BASIC command]
CALL 0F050H

Description: To initialize the LCD, the preassigned I²C slave address 42H is used as an output port. The initialization data is sent to the output port to place the LCD in nybble mode with a 4 × 7 character matrix and invisible cursor. The clear display and home cursor function is called to complete the initialization. Clearing the display and homing the cursor can be used any time after the LCD has been initialized. Since the LCD does not clear from the end of a print string to the end of the LCD line, you will find this function call necessary to keep the screen clean.

Displaying a string is easy without console redirection. The first string, $(0), can be displayed on the LCD by using a simple function call. This displays all characters in the string (normally unprintable characters may be displayed as Kana characters).

Alternatively, and much easier to use, the console ouput device can be hooked in as the secondary or alternate output device. The use of the BASIC UI1 command redirects all output automatically to the LCD display. Once hooked, characters are printed on a character by character basis. Characters between 20H and 7FH are displayed. Those above 7FH are used as cursor control. A <cr> moves the cursor to the begining of the next (or first) line. Using the hook function requires MTOP to be set to 3FFFH (although the function call sets this, the user should be aware that any variable used prior to this call is destroyed unless MTOP is preset to 3FFFH at the beginning of a program).

Custom console output vector locations (4030H–4032H) and custom list@/print@ vector locations (403CH–403EH) are set up in RAM.

NOTE: LCD output routines are considerably slower than console output, therefore care must be taken when using redirected (UO1) output to the LCD while using the BASIC-1 interpreter's INPUT $(0) command. Input characters can be lost while the previous character's ECHO is being displayed when you run above 4800 bps. You can either use a data rate of less than 9600 or redirect console output to the primary (serial port) until after the INPUT statement. Then, when input is complete, redirect the console output to the LCD and PRINT $(0).

Related Topics: I²C BYTE transfers, I²C Keypad, UI0 and UI1 (BASIC commands)

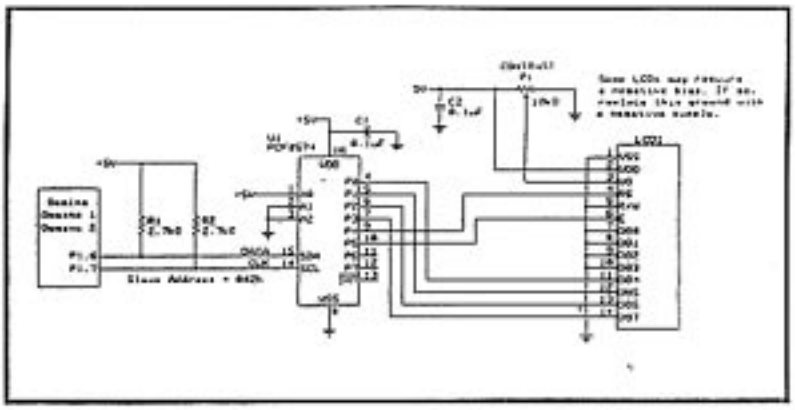
Error Presentation: none

Example:
[direct string output to the LCD]

```
10   MTOP=03FFFH
20   STRING $2,80
30   PRINT"Initialize the LCD"
40   CALL 0F030H
50   PRINT"Now all input strings will be displayed
      on the LCD"
60   INPUT $(0)
70   CALL 0F040H: REM PRINT $(0) TO LCD
80   GOTO 60
```

[redirecting the PRINT command to the LCD as the secondary console output device]

```
10   MTOP=03FFFH
20   STRING $2,80
30   PRINT"Initialize the LCD"
40   CALL 0F030H
50   PRINT"Hook into the secondary console output
      (LCD)"
60   CALL 0F050H
70   PRINT"Now all further output will be displayed
      on the LCD"
80   INPUT $(0)
90   UO1: REM CHANGE TO SECONDARY CONSOLE OUTPUT
      DEVICE
100  PRINT $(0)
110  UO0: REM CHANGE TO PRIMARY CONSOLE OUTPUT
      DEVICE
120  GOTO 70
```



4.7 UTILITIES VERSION NUMBER

Syntax: CALL (address)

Function: The CALL initiates a sign-on message displaying the version number of the utilities presently installed.

Mode: Command, Run

Use: CALL 0FFF0H

Description: Version identification, which is embedded in the utilities, is sent to the active console output.

Related Topics: none

Error Presentation: none

Example:
```
READY
>CALL 0FFF0H
```

Program Output:
```
DOMINO FLASH EXTENSION x.xx IS RESIDENT
```

5.0 CONTROLLING I/O BITS DIRECTLY

Since it isn't possible to directly set or reset bits on Port 3 from BASIC-1, it is necessary to call short machine language routines to do the job. The routines consist of three bytes. The first is either a SETB instruction (D2) or a CLR instruction (C2). The second specifies a bit address. Finally, the third is a RET instruction (22).

The following table details the necessary routines for each of the Port 3 bits. The program example shows how to insert the routines into memory from BASIC-1 and how to call them.

Pin	Bit	Name	To Set	To Clear
P3.0	B0	RxD	D2 B0 22	C2 B0 22
P3.1	B1	TxD	D2 B1 22	C2 B1 22
P3.2	B2	Int 0	D2 B2 22	C2 B2 22
P3.3	B3	Int 1	D2 B3 22	C2 B3 22
P3.4	B4	T0	D2 B4 22	C2 B4 22
P3.5	B5	T1	D2 B5 22	C2 B5 22

Example:
The following code is an example for using INT 1 (P3.3) as an output bit.

```
100     MTOP=31000: REM Set MTOP Lower
110     XBY (32000)=002H: XBY(32001)=083h:
         XBY(32002)=022H
120     REM Put INT 1 Set Program at 32000
150     XBY(32100)=0C2H: XBY(32101)=0B3H:
         XBY(32102)=022H
160     REM Put INT 1 Reset Program at 32100
200     Call 32000: REM INT 1 On
210     Call 32100: REM INT 1 Off
222     Goto 200
```

Note: RAM locations and assembly code can be expressed in decimal, hex, or both.

6.0 UPDATING THE DOMINO UTILITIES

The utilities reside in the uppermost portion of the memory map. The utilities are placed in nonvolatile memory so they remain along with your saved autostarting BASIC program, even after power has become disconnected. The user can take advantage of updated utilities (when available) by simply reloading them. This is a two-step process.

First, a utilities loader program (LOADUTIL.BAS) is entered into DOMINO. When this program is run, you are prompted to download the actual utilities hex file (UTIL_*xxx*.HEX). The LOADUTIL.BAS program reads in each paragraph of the hex file, converting and storing it in RAM. When the hex file has been read, "load sucessful," "call address = xxxx," and "total checksum = xxxxx" messages are displayed. The total checksum should match the one included in the UTIL_100.DOC file. This file also contains any last-minute information you should be aware.

Second, if you have verified that all is correct, you may transfer the utilities using the direct command "CALL xxxx" as displayed in the above message. You get transfer status and a sign-on banner when the utilities have been transfered into nonvolatile EEPROM.

7.0 GETTING STARTED

Although you can use any communication software with DOMINO, Host-1 is a convenient and friendly interface between your PC and DOMINO. Host-1 can be used on any DOS-compatible PC with 640 KB of conventional memory. To use Host-1 with DOMINO, you need two serial ports. COM1 for the serial connection to DOMINO and COM2 for your serial mouse. (If you use COM1 for your serial mouse, you may select an alternate COM port for the DOMINO through the Serial Option of the Main Menu.)

Connect the DOMINO hardware to the PC's serial port and turn on the power to the DOMINO. At this point, unless you already have an autostart program in DOMINO, it waits to receive a space character. (If you are using a simple comm program like Procomm to communicate with DOMINO, remember that DOMINO sets the baud rate when a space character is entered. Any other entry confuses DOMINO. You also need to power DOMINO off and on again if the first character DOMINO receives is NOT a space.)

From the DOS command line, type in Host-1 from the installed directory. Host-1 sets up the screen into windows. The top window is the editing window where you input and revise your programs. The middle window is the console output window where you see DOMINO's output. The narrow bottom window is the console input window where you can type direct commands to DOMINO. You can activate either the editor (top window) or the console (bottom two windows) by clicking on them with the mouse.

Host-1 automatically sends out a space character in an attempt to make contact with the DOMINO. You receive an OK message if all is well.

Click on the top window. Host-1 automatically numbers your BASIC program's lines. Enter this single line where Host-1 has entered the line number 10.

```
10   PRINT"Hello World"
```

Now click on the console window, then the PROGRAM item of the menu bar, and then on SEND ALL. Host-1 sends the BASIC code from its editor to DOMINO.

Click on the RUN item on the menu bar and then on START. Host-1 passes the run command to DOMINO and your program executes (out of RAM).

```
Hello World

READY
>
```

You can start the program from the console input window by typing:

```
RUN<cr>

Hello World

READY
>
```

This can be saved as an autostart program by typing:

```
CALL 0FF20H<cr>

SAVING PROGRAM...
PROGRAM SAVED!
```

Remove power from the DOMINO and then power it back up. The program automatically runs.

```
Hello World

READY
>
```

Please read the *Host-1 Develop System for BASIC-1 CPUs* for complete information on using Host-1, the *BASIC-1 Programming* for more on BASIC's command syntax, and this manual for more on using the DOMINO Utilities.

Devices sold by Micromint are covered by the warranty and patent indemnification provisions appearing in its Terms of Sale only. Micromint makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. Micromint makes no warranty of merchantability or fitness for any purposes. Micromint reserves the right to discontinue production and change specifications and prices any time and without notice. This product is intended for use in normal commercial applications. Applications requiring extended temperature and unusual environmental requirements, or applications requiring high reliability, such as military, medical life support or life-sustaining equipment, are specifically **not** recommended without additional processing by Micromint for such application.

APPENDIX 1.0

1.1 SAMPLE APPLICATION: COMMUNICATIONS

DOMINO-1 can communicate with other serial devices at up to 19,200 bps. It can be connected in one of three configurations: RS-232A, RS-422, or RS-485. DOMINO-1's RS-232A output can be used with most full-duplex PC-type serial devices which normally handle RS-232C provided they can reconcile receiving the lower-voltage transmit level of RS-232A. This three-wire (Tx/Rx/GND) RS-232A connection is created by using the RS-422 input receivers as simple level-shifting inverters as shown in Figure 1.

RS-422 is an alternate full-duplex connection which uses two twisted-pair transmission lines (i.e., Tx+/Tx–/Rx+/Rx–) offering long transmission paths and noise-cancelling techniques. This distance is typically 4000 feet. This connection is shown in Figure 2.

RS-485 is similar to RS-422 with the exception that it uses a single twisted pair in a half-duplex arrangement (i.e., +/-). This means data transmissions must use the same twisted-pair path to travel in both directions, requiring a simple protocol of only one unit seizing the transmission pair at a time while all others listen. This connection is shown in Figure 3.

Figure 1—Typical RS-232A connections

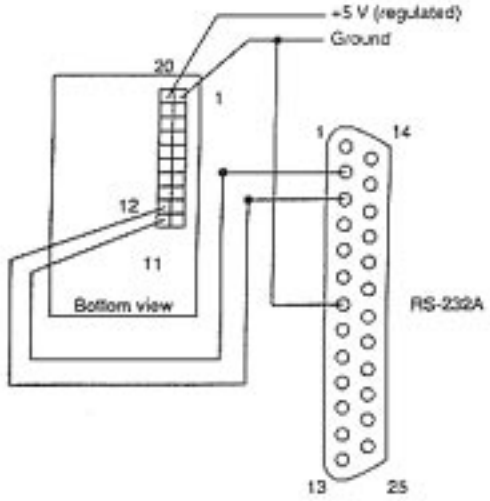
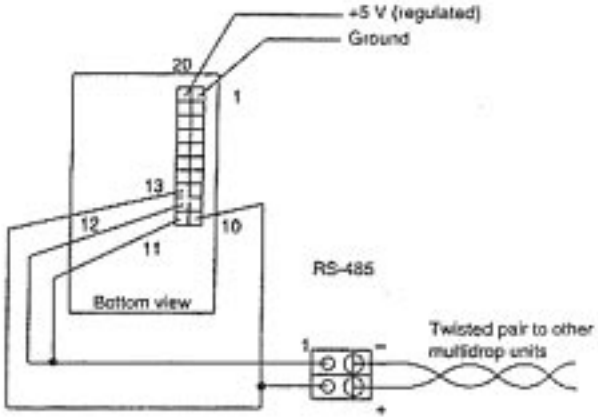
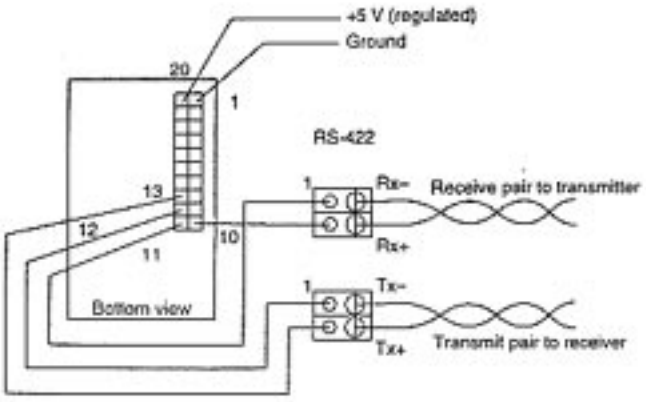


Figure 3—Typical RS-485 connections



Note: RS-485 requires master-slave protocol and direction control

Figure 2—Typical RS-422 connections



1.2 SAMPLE APPLICATION: ANALOG INPUT MEASUREMENT

The DOMINO-1A contains an optional 2-channel, 12-bit A/D converter. The converter is a Linear Technology LTC1298. It is mounted internally in the 52A version, but can be externally connected to a regular parallel I/O model DOMINO-1. Both DOMINO-1 and DOMINO-1A firmware support ADC calls for 8-bit (ADC0832) and 12-bit (LTC1298) dual-channel ADC devices.

In both applications, the ADC chip is connected to P1.7, P1.6, and P1.5 as described in the pinout listing. When an ADC is connected, these port lines shared functions. The user must take care not to confuse functions with random outputs to these lines.

The example below shows ADC connections for using DOMINO-1A with the optional internal ADC or DOMINO-1 with an external ADC attached.

Figure 1—Typical connections for using DOMINO-1 with a user-supplied external ADC

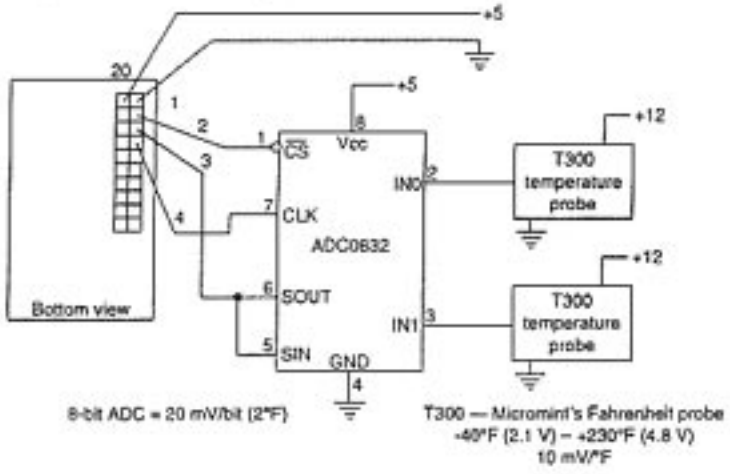
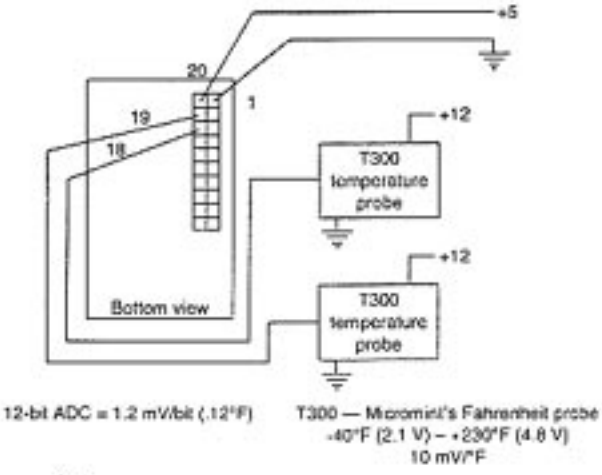


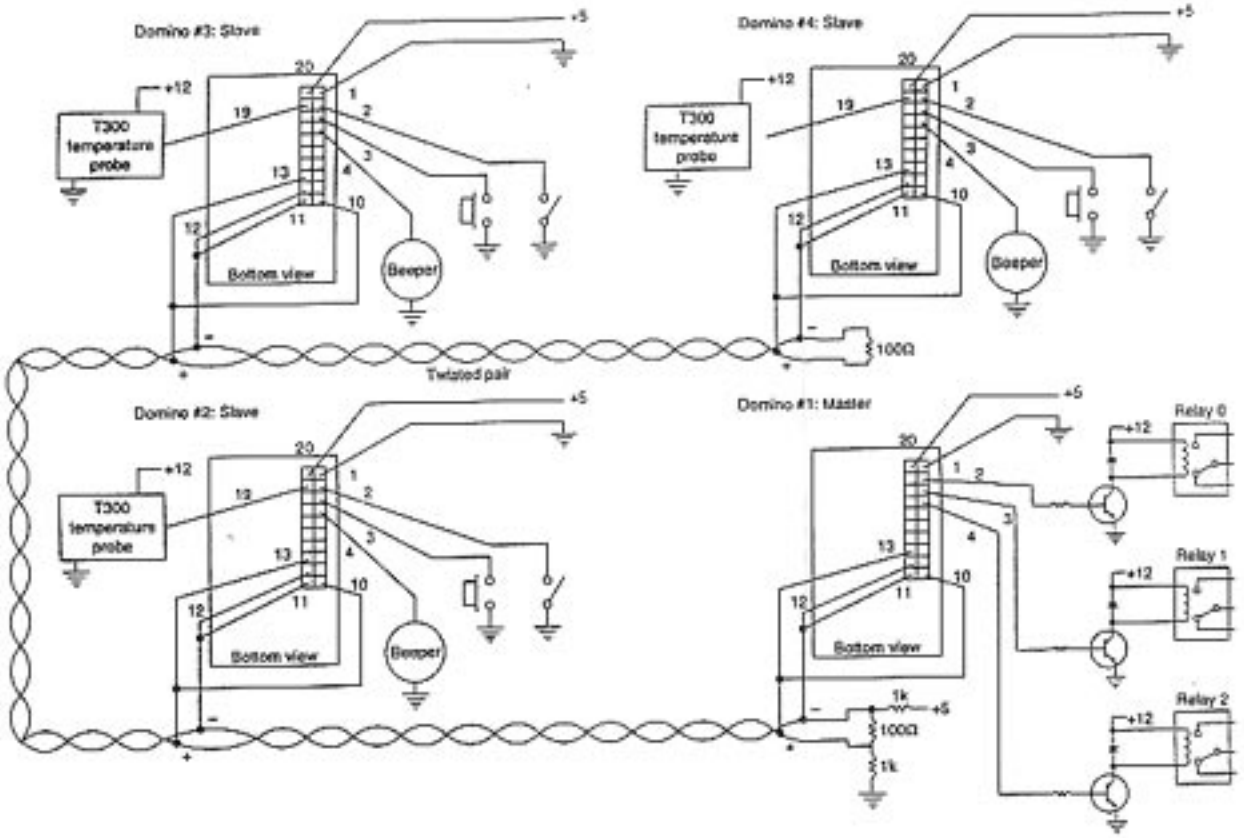
Figure 2—Typical connections for using DOMINO-1A with its internal ADC



1.3 SAMPLE APPLICATION: NETWORKING DOMINO-1

Multiple DOMINO-1s can be used in a networked multidrop configuration using only a single twisted pair for communication. Network protocol requires that only one unit is allowed to transmit on the line at a time. All other units are "listening" in receive mode.

This is accomplished by requiring one DOMINO-1 or a device like a PC to be the net master. The master talks to any slave unit either passing information to it or requesting information from it. The slaves must never answer the master until a response is requested. The master then relinquishes the net to that slave for the response and regains the net when the slave is finished. This arrangement enables multiple controllers to work together gathering numerous inputs and controlling innumerable outputs, independent of the system's size.



APPENDIX 2.0

2.0 DOMINO DEVELOPMENT BOARD

Packing so much power into DOMINO's tiny package really keeps your finished product small and light weight. This may, however, present a problem in the development phase of your product. Micromint offers the DOMINO Development Board as a tool to help you reach your goal in the shortest possible time.

The development board offers regulated 5-V power, communication connection, an external LTC1298 ADC (for use with DOMINO-1's without internal ADC, and a prototyping area. Simply add a regulated or unregulated power supply (7.5–12 V) and an RS-232 cable (DE-9M/DE-9M) to your PC's serial port. Any generic communications program may be used to talk to DOMINO (pressing the space bar as the first character sent enables DOMINO to automatically detect the baud rate in the range of 300–19,200). Micromint offers Host-1, a combination editor and communications program specifically designed for use with 80C52 BASIC systems.

2.1 DOMINO DEVELOPMENT BOARD POWER SUPPLY

Domino requires very little current to operate. Any regulated or unregulated 7.5–12 VDC supply of at least 100 mA can be used. The DOMINO development board accepts a 2.5-mm power plug (available on most small wall transformer power supplies). DOMINO's actual requirements are typically less than 15 mA, but you will probably want some additional power for external circuitry.

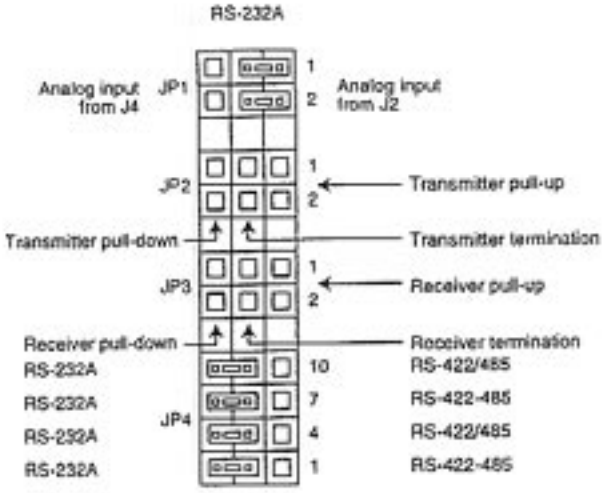
NOTE: Make sure your power supply uses the center conductor of the power plug as ground. Applying reverse voltage to the DOMINO development board damages the regulator and/or the DOMINO module.

2.2 DOMINO DEVELOPMENT BOARD COMMUNICATIONS

2.2.1 RS-232A COMMUNICATIONS

Communications is set for RS-232A from the factory. This setting is necessary for communicating with a PC (using no additional equipment besides a DE-9M to DE-9F cable).

NOTE: RS-232A is characterized as a ±5-V bipolar signal (as opposed to RS-232C at ±12 V). Drivers and receivers are actually RS-422 and the interface is an RS-423 connection (single ended to differential).

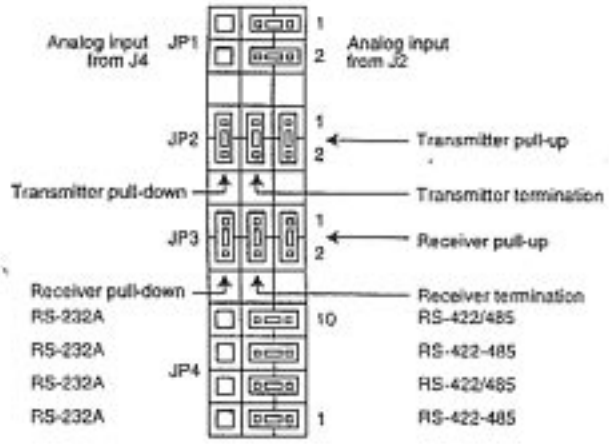


2.2.2 RS-422 COMMUNICATIONS

RS-422 communications requires two twisted pairs. One pair connects the console transmitter to DOMINO's receiver while the second pair connects the console receiver to DOMINO's transmitter.

RS-422 uses two unidirectional data paths—one path for each direction. The data transmission is differential, enabling the noise picked up on the pairs to cancel itself out. Each twisted pair should have termination enabled at each end of the line. Pull-up and pull-down termination may be required, but only at one end of each pair.

RS-422



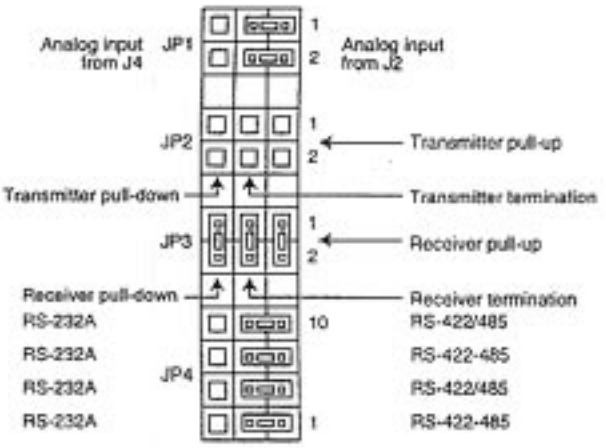
2.2.3 RS-485 COMMUNICATIONS

RS-485 communications requires one twisted pair to connect the console to DOMINO. RS-485 uses one data path, so the drivers at each end must NOT be enabled at the same time. The user is responsible for this rule NOT being broken.

The easiest protocol to follow is a master/slave(s) relationship, where the slaves DO NOT enable their transmitter (respond) unless the master asks them to. The data transmission is differential allowing picked up noise to cancel itself out.

The twisted pair should have termination enabled at each end of the line. Pull-up and pull-down termination may be required, but only at one end of the pair.

RS-485



2.2.4 COMMUNICATION CONNECTIONS

RS-232A connection is made using a DE-9F to DE-9M cable between the PC's serial port and the DE-9F (J3) on the DOMINO development board.

RS-422 connections are twisted wire pairs connected to each set of screw terminal blocks on the DOMIMO development board. Connect the console's transmitter (+) to DOMINO's screw terminal Rx (+) at T1 and the console's transmitter (–) to DOMINO's screw terminal Rx (–) at T1. Connect the console's receiver (+) to DOMINO's screw terminal Tx (+) at T2 and the console's receiver (–) to DOMINO's screw terminal Tx (–) at T1.

RS-485 connections are a single twisted pair connected to both sets of screw terminal blocks on the DOMINO development board. Connect the twisted wire's (+) lead to both of DOMINO's Tx and Rx (+) screw terminals on T1 and T2. Connect the twisted wire's (–) lead to both of DOMINO's Tx and Rx (–) screw terminals on T1 and T2.

2.3 DOMINO CONNECTIONS

Except for communications connections, all pins on the DOMINO are brought out to connector J2. If a 2 x 10 square-pin header is used in J2, a ribbon cable plugged onto J2 has the same signal pinout as if it were plugged onto DOMINO directly. This enables any external circuitry you've developed for DOMINO to be used along with the DOMINO development board.

If you choose to use the prototyping area on the DOMINO development board, you can access to DOMINO's I/O signal at J2.

Analog input signals can come in through J2 or the analog input connector J4. Choose the appropriate input path using JP1. For jumpers toward the edge of the board on JP1, select analog input from connector J2. For jumpers away from the edge of the board on JP1, select analog input from connector J4.

Jumper J5 offers connection between the development board's 5-V power and the power pin on J2. You may wish to run your external circuitry from the development board's power or vice versa. Power is NOT connected between the two systems unless you determine it necessary.

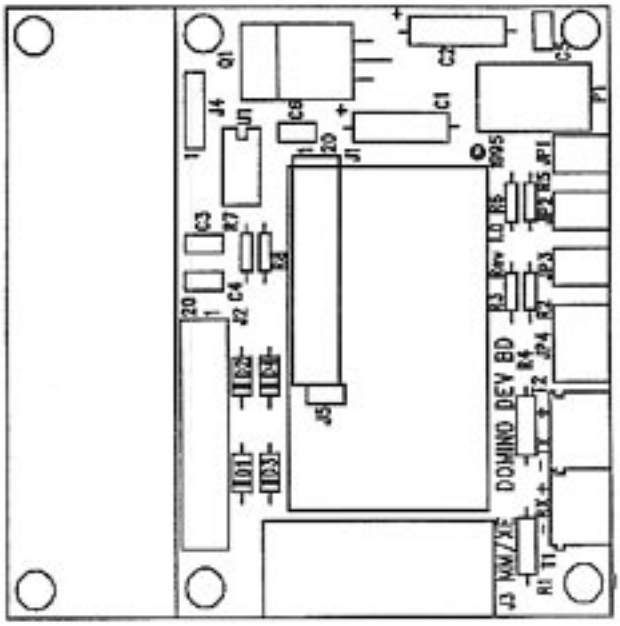
2.4 ANALOG-TO-DIGITAL CONVERTER

DOMINO-1A contains a 2-channel 12-bit A/D converter. For those of you who have purchased a DOMINO-1 without the internal ADC, the DOMINO development board gives you access to an external 2-channel 12-bit ADC using an LTC1298. The utility routines within DOMINO can access this external ADC as if it were installed internally.

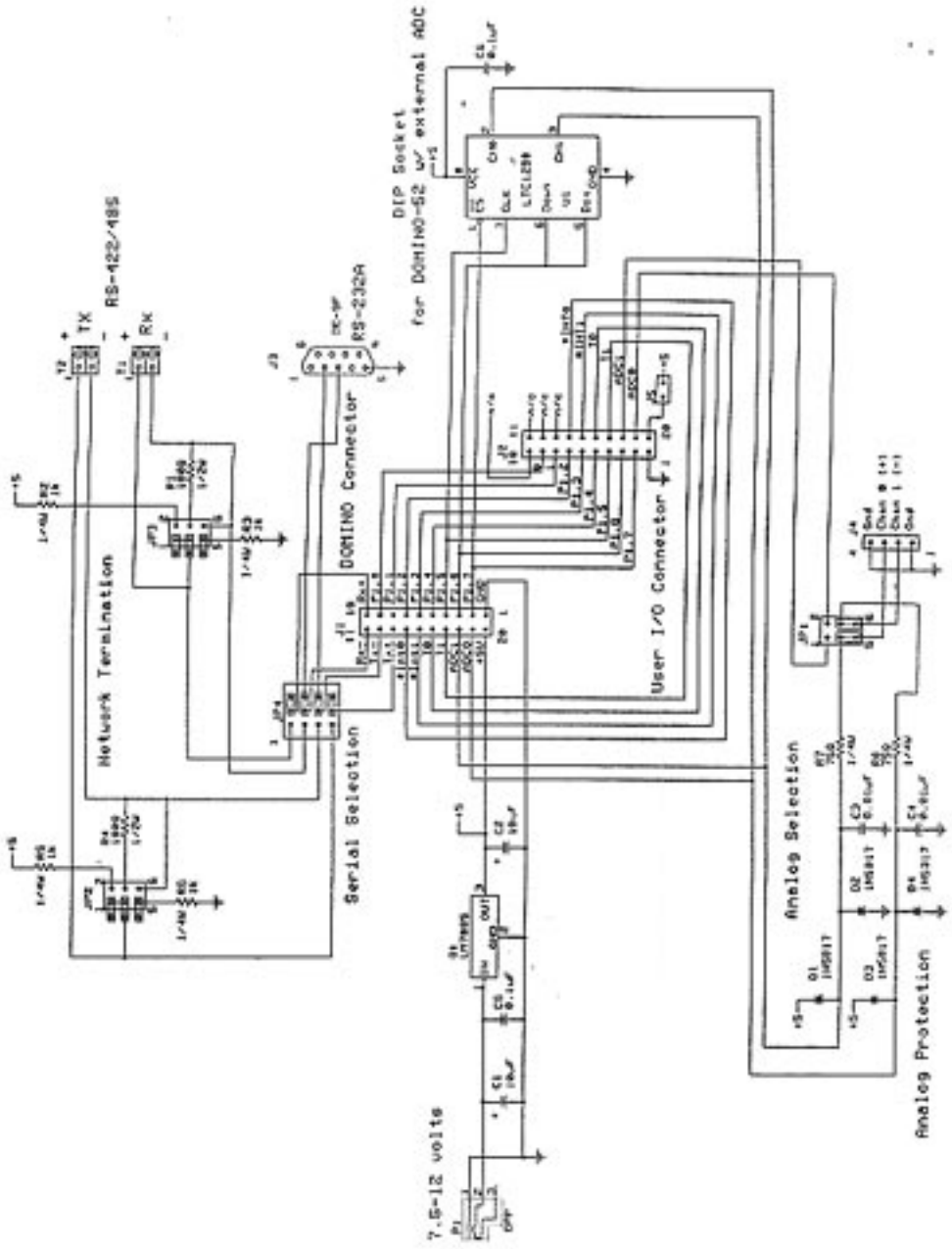
The ADC inputs on the DOMINO development board are available at two locations. The actual input from J2 or J4 is selected through JP1. Input protection is installed on the ADC inputs consisting of a low-pass filter and protection diodes to VCC and ground.

NOTE: When using a DOMINO with internal ADC, please remove any external ADC.

Here's a silkscreen of Domino's Development Board. The development board is used to make it easy to connect the Domino module to external devices during product development.The schematic is on page 22.



SCHEMATIC FOR DOMINO-1 DEVELOPMENT BOARD



USING ASSEMBLY LANGUAGE PROGRAMS WITH DOMINO-52
Utilities Version 1.3 DOMINO-52

The following is a copy of the README.TXT file from the OM-UTL disk. This disk is included with the Domino Development ystem and is not generally sold separately. If you wish to use ny of the features described below you must use the utilities on his disk. For further information on obtaining these utilities, ontact the Micromint sales department at (800)635-3355.

DOMINO-52 has become very popular in the world of embedded ontrol. This is primarily due to the programming simplicity of OMINO's onboard BASIC interpreter. DOMINO's user friendly pro-. ramming environment gets the application up and running in the hortest time possible. Finally, because the application can be tored as a non-volatile auto-starting program directly in the nboard EEROM, virtually nothing is required to go from a develop- ent environment to turnkey production use.

For the most part we have succeeded in providing the proper ombination of high-level language performance and ease of devel- pment and use. If users have ever expressed a wishlist of mprovements, it has only been an interest in making it easier to ncorporate and call assembly language programs along with BASIC.

From day one, of course, DOMINO has had the capability of alling an assembly language program. To get the program into mem- ry requires entering the program via BASIC DATA statements and oking the program into a specific memory address. Such a proce- ure is acceptable for short programs.

As users apply DOMINO to increasingly sophisticated applica- ions, the demand to go beyond BASIC and a few small assembly rou- ines has also increased. In Version 1.3 of the DOMINO utilities, e have tried to address the wishes of those who want to use arger assembly language programs. Of course, given the physical ardware, not everything is possible.

For a DOMINO-52 with V1.3 utilities (all DOMINOs can be pgraded to the latest utility version), the programming environ- ent can be thought of as 4 different modes: BASIC only; BASIC ith callable assembly routine(s) entered via DATA statements; ASIC with EEROM variable storage; and, BASIC with callable small r large assembly routine(s) entered via a HEX download. The lat- er 2 modes are new with version 1.3. The major difference fforded with V1.3 is that the entire contents of SRAM is now tored to EEROM, not just the BASIC program. If that SRAM contains large assembly language program, both the BASIC and that program ill be saved. Please note that executing an assembly language rogram saved by that method requires the user to follow a few ules regarding the ORG addresses. More on this later.

First, here is a description of how we perceive each of these odes:

ASIC ONLY

DOMINO-52 was designed for use with the internally masked BASIC nterpreter. For most applications the BASIC language will be ufficiently powerful.

ASIC W/INCLUDED ROUTINE IN DATA STATEMENTS

Advanced programmers might wish to include an assembly language outine to achieve increased performance within a particular pplication program. A CALL from BASIC can be made to either a ompiled or assembled routine. Small routines can be poked into rotected SRAM using DATA statements. This approach keeps all the ode, BASIC and assembly, in one easily maintainable file. The ollowing illustrates poking an assembly language program to toggle utput bit P3.5 when called:

```
0 MTOP=1FFFH                    : REM PROTECT MEMORY ABOVE 1FFFH
0 FOR X=2000H to 2005H          : REM ADDRESSES TO PUT DATA
0 READ D                        : REM READ A BYTE OF DATA
0 XBY(X)=D                      : STORE THE DATA BYTE AT ADDRESS x
0 NEXT X                        : REM DO MORE
0 REM YOUR PROGRAM GOES HERE


00 CALL 2000H : REM THIS CALL SETS OUTPUT BIT P3.5 (T1)


00 CALL 2003H : REM THIS CALL CLEARS OUTPUT BIT P3.5 (T1)


99 END
000 DATA 0D2H, 0B5H             : REM SETB P3.5
010 DATA 022H                   : REM RETURN
020 DATA 0C2H, 0B5H             : REM CLR P3.5
030 DATA 022H                   : REM RETURN
```

ASIC AND LARGE ASSEMBLY CODE

At some point the assembly language routine becomes too large to nclude as DATA statements. The STOREHEX.BAS program loads your ompiled or assembled .HEX file into SRAM for a debugging session if t was ORG'd for the SRAM addresses between 2000H and 6FFFH. Or, if t was ORG'd for the EEROM (between 9000H and 0DFFFH) STOREHEX.BAS ill relocate the load properly into SRAM so it will be saved along ith your BASIC program when using the BASIC program save command PROG' (utilities V1.3).

Placing your routine in SRAM for debugging:

Assemble the routine using an ORG=2000H (or higher but not extending beyond 6FFFH.)
Set MTOP to below ORG address to protect upper SRAM MTOP=1FFFH (Remember to leave room for your BASIC code).
Use STOREHEX.BAS to load in yourfile.HEX for SRAM.
Use NEW command to delete STOREHEX.BAS
Load in your BASIC application.
NOTE: BASIC program must fit below MTOP. Assembled routine must not extend beyond 6FFFH.
Test application.

Placing your routine in SRAM for saving into EEROM:

Assemble the routine using an ORG=9000H (or higher but not extending beyond 0DFFFH.)
Set MTOP as previous to protect upper SRAM. MTOP=1FFFH (Remember to leave room for your BASIC code).
Use STOREHEX.BAS to load in yourfile.HEX for EEROM.
Use NEW command to delete STOREHEX.BAS
Remember to change the CALL statements in your BASIC code to reflect addresses that changed in your assembled routines.
Load in your BASIC application.
NOTE: BASIC program must fit below MTOP. Assembled routine must not extend beyond 0DFFFH.
Save the BASIC program (and your routine) using one of the PROG calls (FF00h-FF20h). Remember to use the appropriate PROGx command if you wish to have the program auto-start.

SAVING CONFIGURATION VARIABLES

If your program needs to update some configuration information as part of the executing application, refer to SAVECFG.BAS program. You may include the appropriate parts of this program which allows up to 60 floating point numbers to be saved into 10 protected blocks of EEROM. 6 floating point numbers are stored into each 64 byte block using a storage routine loaded from DATA statements. The DATA statements are XBY'd into SRAM at 7000H and must execute from this location. A 64 byte block of SRAM is set aside at location 7FC0H-7FFFH. This PAGE is used as a transfer buffer for any variables stored by the BASIC application. A CALL 7000H unlocks and transfers this block of DATA into the EEROM. The user is responsible for placing the data into the block prior to calling the routine with the PAGE number (0-9) at which to store it within the EEROM. Although once stored the DATA can be retrieved (read) directly from the EEROM (0F800H-0F83FH for PAGE 0) you may wish to move it down to the 64 byte SRAM block to keep the bookkeeping straight.

Please keep in mind that unlike SRAM, EEROM can only be written to a finite number of times (like SRAM, however, it can be read from an infinite amount). While this number of writes is typically many thousands of times, a program routine which constantly reads something like an input variable and stores it to EEROM directly will make short order of the EEROM. DO NOT USE THE EEROM FOR DIRECT STORAGE DATA LOGGING. If you must save something when the power is off, either use the configuration variable storage technique described above or only move the logged data from SRAM to EEROM just prior to power down.

Each DATA statement in the BASIC program is one assembly language instruction. Line 1200 and 1210 actually set the 10 page beginning address. BE CAREFUL IF YOU MAKE ANY CHANGES, A SIMPLE ERROR MAY CAUSE PERMANENT AND IRREVERSIBLE DAMAGE to DOMINO, VOIDING ANY WARRANTY.

FILES ON THIS DISKETTE

UTIL_130.HEX	INTEL.HEX file of the Version 1.3 Utilities (use LOAD_130.BAS to reload into DOMINO)
LOAD_130.BAS	Program written in BASIC to reload the utilities (UTIL_130.HEX) into DOMINO
UTIL_130.DOC	Revision summary and information on reloading the utilities into DOMINO
STOREHEX.BAS	Program written in BASIC to load a compiled/ assembled routine into SRAM for debugging or temporarily into SRAM for storage to the EEROM
SAVECFG.BAS	Program written in BASIC which demonstrates how floating point configuration values may be stored and updated to/from EEROM
README.TXT	This file

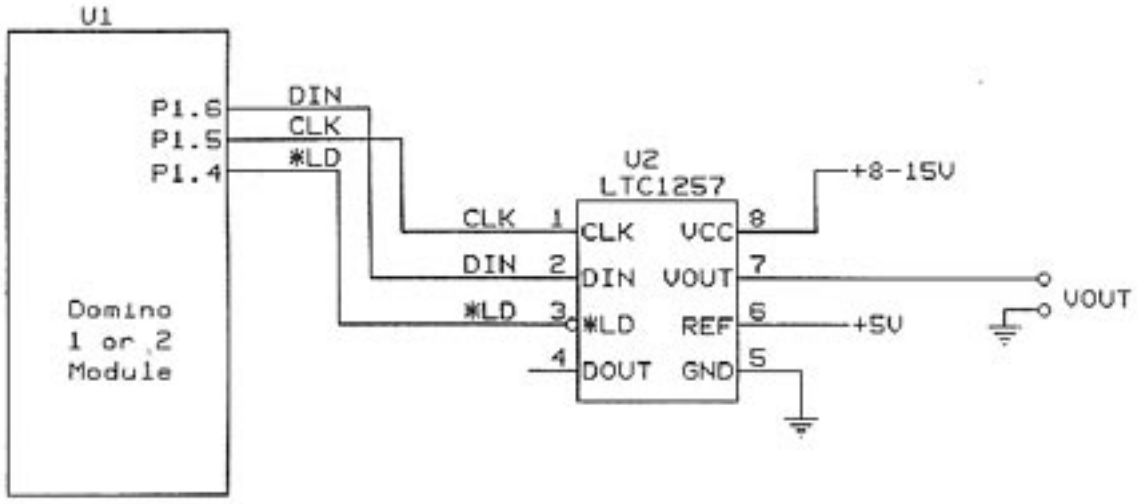
Application Note

Adding a DAC to the Domino – Connecting an LTC1257 2-channel, 12-bit DAC

Date: 3/6/98 | By: J. Gorsky

Even though the Domino modules offer a wide range of features, sometimes you need a little something extra like a DAC output. Connecting a serial device such as the Linear Technologies LTC1257 DAC can be accomplished easily with a few lines of BASIC code and three bits of Port 1.

The sample code and schematic show how this is done. If additional outputs are needed, the clock and data lines may be shared between devices and additional Port 1 pins used to select the individual chips.



Sample BASIC Code

```
10    REM Domino - LTC1257 DAC Program
20    REM *LD = Port1.4, CLK = Port1.5, DIN = Port1.6
30    LD = 10h : CK = 20h : DA = 40h
40    PORT1 = PORT1 .OR. LD : REM raise *LD
50    PORT1 = PORT1 .AND. (0FFh-CK) : REM drop CLK
60    PORT1 = PORT1 .AND. (0FFh-DA) : REM drop DIN
70    INPUT "Enter desired output voltage: ",V
80    V = V/(5/4095)
90    FOR X = 11 TO 0 STEP -1
95    REM TEST VALUE, IF BIT=1 THEN  CLOCK A 1, ELSE  CLOCK A 0
100   IF (V.AND.2**X)<>0 THEN  GOSUB 2000 ELSE  GOSUB 1000
110   NEXT X
120   GOSUB 3000
130   PRINT
140   GOTO 70
1000  PORT1 = PORT1 .AND. (0FFh-CK) : REM drop CLK
1010  PORT1 = PORT1 .AND. (0FFh-DA) : REM drop Din
1020  PORT1 = PORT1 .OR. CK : REM raise CLK
1030  RETURN
2000  PORT1 = PORT1 .AND. (0FFh-CK) : REM drop CLK
2010  PORT1 = PORT1 .OR. DA : REM raise Din
2020  PORT1 = PORT1 .OR. CK : REM raise CLK
2030  RETURN
3000  PORT1 = PORT1 .AND. (0FFh-LD) : REM drop *LD
3010  PORT1 = PORT1 .OR. LD : REM raise *LD
3020  RETURN
```

Micromint
Chips and Modules

Product Use and Handling

The contents of this package are sensitive electronic components and must be used and handled with great care to avoid damage. Please note and observe the following items to prevent electrical and/or mechanical damage that might cause the product to become inoperable or unusable or void the warranty.

TESTING

Each product is tested at the factory and is known to be working at the time of shipment to you. Please handle accordingly.

PACKING

These products have been inserted into antistatic material to prevent static discharge and lead damage. The units are securely held by the package design, and it is recommended that the package be saved should return shipment be required for any reason.

Units returned that have been damaged by shipping or handling will not be covered under warranty.

STATIC DISCHARGE

These products are manufactured using high-density MOS integrated circuits technology and, consequently, may be susceptible to damage from Electrostatic Discharge (ESD). To avoid damage, do not touch the pins unless you are grounded in a safe manner or you are in an otherwise antistatic environment.

INCOMING INSPECTION

Be careful not to insert the product in a socket backwards. The corner location of Pin 1 on the device is indicated by a label on the top or bottom of the module and a dot or notch at one corner on the top of the device. Improper insertion may destroy the product on application of power.

Micromint recommends installing module and chip products into socket strips or a connector end after incoming inspection and acceptance.

Products damaged by user manufacturing processes are rendered unsalable and will not be accepted as valid returns.

NO-CONNECT PINS

Pins marked N/C (No Connection) should not be connected. They should be left floating. These pins may be used as test points during the manufacturing process and may be damaged if connected to anything.

PRODUCT OPERATION

Please refer to the applicable product specifications and application notes before the application of power and/or signals to these products.

Careful adherence to these precautions should result in damage-free operation of the product.

Micromint, Inc.
4 Park St.
Vernon, CT 06066

(860) 871-6170
Fax: (860) 872-2204
BBS: (860) 871-1988

http://www.micromint.com/

© 1996, Micromint, Inc.